The Untitled Raccoon Game

The 3D models, effects and game components are finished and working. We implemented a world with 2 levels, a 2 part gameplay system with a building phase and a running phase, usable objects for building, a goal and a playable and animated raccoon character.

Implementation:

Camera system (Intuitive Camera):

Since we have a 2 part gameplay system with 2 completely different views and movements we thought it to be best to implement 2 different camera headers: Camera1.h and Camera2.h.

- **Camera1.h** is for our first phase of the game and is similar to a first person view. You can move freely in the world so you can find and gather all objects you will need to build a path.
  Implementation: Similar to Framework from ECG. Only the rotations are multiplied first and then the translation so the camera seems to be in first Person.

- **Camera2.h** then is for the 2nd phase of our game and locked onto our main character the raccoon to give you a 3rd person view onto our world. In this part you thus play as the raccoon to try and make your way up to the goal.
  Implementation: Similar to Camera1 but the position is always a specific distance behind the character with the same yaw as the character.

Model loading (3D Geometry, Textures):

To load all of our models(or levels) and not clutter up our code too much with repetitive calls we decided it would be best to make a new header called Level.h and make use of a .ini file to be able to load the levels as a whole.

- The **.ini file** contains the object's path as well as necessary transformations.

- **Level.h** helps us load and assign and transform the Objects referenced in the .ini file.
  Implementation: Models are implemented through the implementation of assimp and uses other header files such as **Border.h**, **Light.h**, **World.h**, **Hitbox.h**, **KomplexStaticBody.h** and **StaticHitbox.h** to further structure our code.

Basic and Advanced Physics (Moving Objects, Collision Detection):

For either basic and advanced physics we first had to compile and add Physx to our Framework. Physx is then used to build Hitboxes with Collisions.

Implementation: Every object is either a PxStaticBody (which are called static-objects) or a PxRigidBody (which are called dynamic-objects) with a HItbox similar to the .obj model but in a simpler form like a box or pill. Static-objects are objects which can't be moved but you can stand on them or get blocked by them. Dynamic-Objects can be slightly moved by the character and they can fall and interact with each other.
With the left mouse button pressed, a ray gets cast from the camera with which you can select a dynamic-object to drag and pull along the mouse cursor.

User Interaction (Gameplay, Intuitive Controls, Win/Lose Condition, Min. 60 FPS and Frame Rate Independence):

Main features of the gameplay are implemented and the game can be played and navigated from within the running game. The controls are night still change a little bit when it comes to their impacts/sensitivity (eg. cursor movement) but are largely finalized. The game can be exited at any time or stop when our win condition (the raccoon reaching the food) returns true. The game and thus the gameplay is split up in aforementioned 2 phases:

- Phase 1: controls for 1st player movements → (see "Features" part)
- Phase 2: controls for 3rd player movements →(see "Features" part)

GPU Vertex Skinning:

Skinning and animating our main character the raccoon is important because we don't just control him but we see him running and jumping actively in our 2nd phase and thus he needs to be able to show some kind of running and jumping animation.

Implementation: First the vertex shader had to be adjusted in a way that he also holds the bone ID and the weight for each vertex as well as the maximum amount of bones that can influence a vertex and some changes in the main to apply all transformations to vertex that is influenced by bones. Additionally we needed some code changes in order to actually load and work with the skeleton/bones of the model and implemented animation. With those changes we could then switch to using COLLADA (.dae) files instead of .obj files for our animated character. The static and building objects keep getting loaded from .obj files.

Physically Based Shading (Textures):

To guarantee having a cartoony look to our game but not just have simple flat colored objects we wanted to try ourselves in PBR Shading. This is implemented in our shaders "shader" and called upon in our Main.cpp.

Implementation: If assimp loads multiple textures a boolean is switched so the PBR calculations are used. For these calculations you need a set of 5 different textures (albedo/ambient, roughness/specular, normal, height/ao and metallic) which define the attributes of the fragments.

Shadow Map with PCF (Illumination Model):

We decided that Shadow Maps would provide us with the desired results when it comes to shadow and light as our game will only run at in-game night time. The shadow maps are implemented in separate shaders called “shadowDepthMapping” and “shadowDepthCubeMapping” and then called onto in Main.cpp.

Implementation: “shadowDepthMapping” generates a 2D depth map by looking in the direction of the directional light, from a moving spot, which is moving with the camera. “shadowDepthCubeMapping” generates a cube map by looking in six directions from the lights position.
These maps are then used to see if a fragment is seen by the light and therefore lit or in the shadow. If it’s in the shadow it will only display the ambient light.

“Features” of the game

Phase 1:

- simple HUD to explain game mechanics
- use W,A,S,D key to navigate in the world
- by pressing and holding the right mouse button you can look around
- by pressing and holding the left mouse button you can move a selected objects to a desired location (the cursor has to be on the desired object) place or drop the object by releasing the left mouse button
  Left mouse button + mouse wheel can be used to pull the object towards yourself or to push it away
- Esc key to quit the game
- F3 to switch between day and night time
- Enter key to switch to phase 2

Phase 2:

- use W,A,S,D key to navigate with the raccoon in the world
- Mouse movement to rotate the camera as well as the direction the raccoon looks towards
- Spacebar to jump with the raccoon
- Enter key to switch to phase 2 (or go to next level if level completed)
- Esc key to quit the game
- F3 to switch between day and night time
- Backspace key to reload level

Additional Libraries

- Physx: https://developer.nvidia.com/physx-sdk
- Assimp: https://github.com/assimp/assimp/releases

- Freetype: https://freetype.org/
- Stb: https://github.com/nothings/stb
- Learnopengl: https://learnopengl.com/Code-repository

Additional Used Code

- https://community.khronos.org/t/about-glm-lookat-function/74506
- https://stackoverflow.com/questions/20140711/picking-in-3d-with-ray-tracing-using-ninevehgl-or-opengl-i-phone/20143963#20143963